REMARKS

Claims 1-4, 6-13, 15-21 and 23-37 are pending.

Applicants acknowledge and appreciate that the Examiner has withdrawn certain rejections from the previous Office Action dated June 10, 2010. Applicants also acknowledge and appreciate that the arguments provided by Applicants were deemed persuasive with respect to these rejections; however new grounds of rejections have been provided in the present Office Action in view of U.S. Patent Application Publication US 2005/0114771 ***(Piehler)***.

Claim Rejection – 35 U.S.C. §102

Claims 9-13 and 15 are rejected under 35 U.S.C. 102(b) as being anticipated by U.S. Patent No. 5,193,191 ***(McKeeman)***. Applicants respectfully traverse this rejection.

Claim 9

For ease of illustration, claim 9 is discussed first. Claim 9 recites "wherein the instructions when executed enable the processor to initiate compiling of the file based on determining that the file was modified." In the Office Action, the Examiner argues that ***McKeeman*** teaches this claimed feature because ***McKeeman*** discloses "according to one feature of the invention, those modules 12 which have not been changed or are not dependent upon changed code are not recompiled." *See* Office Action, p.6 (citing ***McKeeman***, col. 5, ll. 21-23). As quoted, ***McKeeman*** teaches that code which is **not** changed/modified is **not** recompiled. That is, ***McKeeman*** discloses that once a section of code is compiled, that code does not have to be compiled again if it has not since been modified. In contrast, claim 9 calls for **initiating compiling** based on determining that the file **was modified**. This is clearly different from the teachings of ***McKeeman***. The Examiner is arguing that *not compiling* code because the code was *not changed* is equivalent to initiating compiling based on code that **has been changed**. *See*

Office Action, pp.3-4, Response to Arguments. Such an argument is based upon a logical fallacy. There is no "implicit" disclosure to support the Examiner's assertion. In other words, in view of the teachings in ***McKeeman***, if code is not compiled because it is not modified, it does not logically follow that code is compiled if the code has been modified. Indeed, ***McKeeman*** makes no such teaching, and this is not surprising because ***McKeeman*** is not concerned with compiling modifying code.

Moreover, ***McKeeman*** actually teaches away from the instant claims because ***McKeeman*** is concerned with compile time efficiency gained by *not* having to re-compile unchanged code. *See, e.g.*, ***McKeeman***, col. 5, ll. 21-23; col. 11, ll. 42-61. As such, ***McKeeman*** does not, and cannot, teach **initiating compiling** based on determining that the file **was modified**, as recited in claim 9. That is, ***McKeeman*** is actually teaching the opposite of initiating compiling in the manner called for in claim 9.

For at least these reasons, claim 9 and its dependent claims are allowable. For similar reasons, the remaining claims are also allowable.

For ease of illustration and organization of arguments, further remarks relevant to claims 9-13 and 15 are made in the claim 1 arguments section below.

Claim Rejection – 35 U.S.C. §103

The Examiner rejects claims 1-4, 6-8, 16-21, 23-29, 31-32 and 35-37 under 35 U.S.C. §103(a) as being unpatentable over ***McKeeman*** in view of "Upgrading Microsoft Visual Basic 6.0 to Microsoft Visual Basic .NET" ***(Robinson)***. Applicants respectfully traverse this rejection.

Claim 1

For ease of illustration, claim 1 is discussed first. Claim 1 calls for initiating compilation of a file in a processor-based system in advance of a request from a user to compile the file and detecting the user request to compile the file. Claim 1 also calls for indicating a status of the

compilation of the file in response to detecting the user request. Initiating compilation of the file includes compiling the file in response to determining that the file has been modified.

The Examiner's rejection of claim 1 is improper because ***McKeeman*** and ***Robinson***, either alone or in combination as cited by the Examiner, fail to teach all of the claimed features. For example, claim 1 calls for initiating compilation of a file in a processor-based system in advance of a request from a user to compile the file. In the Office Action, the Examiner admits that ***McKeeman*** does not teach this feature, but the Examiner still argues that ***Robinson*** teaches initiating compilation in advance of a request. *See* Office Action, p.8 (citing ***Robinson***, p.16), p.2 Response to Arguments Section. The Examiner's attention is respectfully directed to ***Robinson***, p.18, which describes the function of the "compiler" as follows:

> "The result is a true Visual Basic experience, enhanced by background compilation as you type. For example, if you misspell the keyword *Function*, as in
>
> **Funtion myFunction**
>
> as soon as you move off the line, the compiler **parses** it and puts a compiler error in the Task List. It also underlines the word "Funtion" with a blue squiggle indicating the location of the compile error. As soon as you correct the line, the compiler removes the Task List item and erases the underline." (*emphasis added*)

As can be seen, before a user-initiated compile, the "compiler" *parses* text after it is entered into a file. The passage cited by the Examiner, however, does not teach or suggest initiating **compilation** of a file in a processor-based system in advance of a request from a user. ***Robinson*** teaches that a background compiler parses text as it is entered into a file in order to alert the user of syntax errors and the like. In the Office Action, the Examiner states that the provided "compilation errors" are evidence that the compiler has actually compiled code. *See* Office Action, p.2, Response to Arguments. However, were the compiler to actually compile the code at this point, an object file or other such output data would be created. ***Robinson*** does not create an output file/data as a result of any such compilation; ***Robinson*** *parses* the code and *flags potential*

errors therein. The Examiner also states that the use of the word "compiler" in place of "parser" is dispositive of the Examiner's position that actual compilation is taking place. *See id.* Again Applicants respectfully point out that simply because a so-called "background compiler" performs a parsing functionality as code is written, this does *not* teach that actual compiling is taking place in the background. Indeed, as noted herein, the actual compiling in ***Robinson*** does not take place until a user elects to compile. The Examiner also makes reference to "Advanced Basics; Scaling Up: The Very Busy Background Compiler" (***Gertz***) citing this references as further elaborating upon the "compiler" of ***Robinson***. *See* Office Action, p.3, Response to Arguments. Applicants note that ***Gertz*** does not compile until code is "committed" by the user. That is, by committing" the code, the user is requesting the compiler to compile the code. In contrast, claim 1 recites initiating compilation of a file in a processor-based system in advance of a request from a user.

Applicants respectfully submit that the Examiner's reliance on ***Robinson*** is misplaced because the parsing is **not** the same as initiating compiling, as would be known to those skilled in the art. **Compiling, as would be known to those skilled in the art, would at least involve forming object files (or the like) from source code.** As Applicants previously stated, assuming *arguendo* that ***Robinson*** teaches a compiler, the compiler simply performs parsing before a user-initiated compilation. That is, the "compiler" in ***Robinson*** does not *compile* code until after the user initiates a compilation. ***Robinson*** does not disclose, *and the Examiner has not cited, any teaching or suggestion in the cited reference* that parsing done by the compiler, as shown in ***Robinson***, initiates **compilation** in **advance** of a request from a user to compile, as called for in claim 1. As such, ***Robinson*** does not, and cannot, teach this claim feature, and, as admitted by the Examiner, ***McKeeman*** fails to remedy the fundamental deficiencies of ***Robinson***.

Claim 1 also calls for "compiling the file in response to determining that the file has been

modified." In the Office Action, the Examiner cites ***McKeeman***, col. 5, ll. 21-23, as teaching this claimed feature. *See* Office Action, p.6. However, as Applicants have stated in above, ***McKeeman*** compiles when the *user* (developer) decides to compile modified code, **not** in response to a file modification, as called for in claim 1. In the Office Action, the Examiner argues that because unchanged files are not compiled, ***McKeeman*** teaches this claimed feature. The Examiner, however, has not shown how this passage teaches compiling the file in response to determining that the file has been modified, and this is not surprising because ***McKeeman*** teaches compilation occurs when the *user* (developer) decides to compile, **not** in response to a file modification, as called for in claim 1. Additionally, as previously stated, claim 1 recites compiling a file in advance of a request from a user to compile, but in contrast, ***McKeeman*** teaches that actual compiling is performed after a user-initiated compile command. As such ***McKeeman*** does not, and cannot, teach the claimed feature of the "compiling the file in response to determining that the file has been modified," as called for in claim 1. ***Robinson*** fails to remedy this fundamental deficiency as ***Robinson*** is similarly concerned with compiling upon a user's command/input. As discussed above, ***Robinson*** parses, but does not compile, prior to a user's compile command.

In the Office Action, the Examiner states that the plain language of the claim "does not require the indication of a status of preexisting compilation result." *See* Office Action, p.3, Response to Arguments. Applicants respectfully disagree. Claim 1 recites "indicating a status of the compilation of the file in response to detecting the user request," where the compilation is performed in advance of a request from a user to compile. Clearly the "plain language" of the claim requires an indication of a status of preexisting compilation result.

Without using improper hindsight reasoning and using the claim as a roadmap, the person of ordinary skill in the art would have no apparent reason to modify the references to arrive at the

subject matter of claim 1. The Examiner essentially provided a conclusory statement that adding the features of these references together would make for a better product; *i.e.*, the Examiner has simply stated the result of such a combination. *See* Office Action, p.8 (stating that the combination would have been obvious "in order to provide a timely indication of errors as suggested by *Robinson*."). As such, the Examiner has merely stated that such a combination would have been obvious. However, the Examiner has not pointed to any teachings in the cited references that would **motivate** a person of skill in the art to combine the references. In other words, the question that must be addressed includes "***why*** would a person have thought to combine the cited references based on their teachings?", and "***what*** was the need?", not simply "what benefits would result?". There must be some motivation or need as to why a combination would have been obvious at the time of the invention.

Applicants respectfully submit that the Examiner's conclusory statement is motivated by improper hindsight and is without support. Applicants respectfully request that the Examiner provide a motivation to combine/substitute that **does not** rely inherently upon the result of such a combination. In other words, a conclusory statement that that "when coupled [the cited references] would teach the claim limitations" is without proper basis and relies entirely upon the result to provide motivation. Applicants respectfully request the Examiner point to a teaching the cited art that shows **where** and **why** a person of skill in the art would have had a need to combine/substitute. In light of the fact that ***Robinson*** specifically discusses parsing and ***McKeeman*** is not concerned with background compiling, the Examiner must show some need for background compilation, not merely a result-oriented statement. Motivation to combine aside, as discussed above, even if ***McKeeman*** and ***Robinson*** were to be combined, claim 1 as a whole would be untaught and non-obvious over the references.

For at least the aforementioned reasons, claim 1 and its dependent claims are allowable. For at least similar reasons, the remaining independent claims, and their respective dependent claims are also allowable (including claim 9 and its dependent claims).

As such, Applicants request this rejection of claims 1-4, 6-13, 15-21, 23-29 and 31-37 under 35 U.S.C. §103(a) be withdrawn.

Claim 2

Other claims are allowable for additional reasons. For example, claim 2 which depends from method claim 1, recites "wherein initiating compilation of the file comprises compiling the file including one or more code segments to produce an object code file." It should be noted that claim 1 recites initiating compiling in advance of a request from a user to compile, therefore the production of an object code file also occurs in advance of the user request. In the Office Action, the Examiner argues that ***McKeeman*** teaches this claimed feature because ***McKeeman*** teaches that object code tables are output from a compiler. *See* Office Action, p.8 (citing ***McKeeman***, col. 5, ll. 30-34). The passage from ***McKeeman*** relied upon by the Examiner teaches that the output of the compiler (*i.e.*, the object code tables) are produced subsequent to a user-initiated compilation. Applicants respectfully assert that the Examiner has taken the cited passage out of context and improperly applied the passage to the instant claims. Taking the entire passage in proper context, from line 15 to line 34, clearly shows that ***McKeeman*** describes a compilation initiated by a user (developer). *See* ***McKeeman***, col. 5, ll. 15-17. As such, ***McKeeman*** does not, and cannot, teach producing an object code file in advance of a request from a user to compile, as called for in claim 1. ***Robinson*** fails to remedy this fundamental deficiency.

For at least the aforementioned reasons, claim 2 and its dependent claims are also allowable.

Claim 3

Amended claim 3 is discussed next. Claim 3, as amended, depends from method claims 2 and 1, and recites "initiating compilation of a file in a processor-based system in advance of a request from a user to compile the file further comprises compiling the file to completion." That is, when compilation is initiated, the compilation will compile the file all the way through to its completion. The Examiner now argues that ***McKeeman*** fails to teach the claimed feature of compiling in the background or compiling in advance of a request from a user. *See* Office Action, p.5, Response to Arguments. ***McKeeman*** discloses compiling to completion *after* a user request to do so. Claim 3, in contrast, requires compiling the file to completion in advance of a request from a user. Further, ***Robinson*** teaches that a compiler may parse text to flag potential compilation errors as a user edits a file. *See* ***Robinson***, p.18. ***Robinson*** does *not* teach that the file is compiled to completion in advance of a request from a user, as recited in claim 3. ***Robinson*** teaches that potential compilation errors are flagged on a line-by-line basis during editing. Indeed, ***Robinson*** does not teach that the file is compiled in advance of a user request at all. The *compiler* taught in ***Robinson*** parses lines of text and does not compile to completion in advance of a user request, as recited in claim 3. As such, ***Robinson*** does not, and cannot, teach this claimed feature. ***McKeeman*** fails to remedy this fundamental deficiency.

For at least the aforementioned reasons, claims 3 and 32, and their respective dependent claims, are also allowable.

Claim 30

Claim 30 is rejected under 35 U.S.C. 103(a) as being unpatentable over ***McKeeman*** and ***Robinson***, and further above in view of U.S. Pat. Pub. No 2005/0108682 ***(Piehler)***. Applicants respectfully traverse this rejection.

Claim 30 depends indirectly from independent claim 24. Because ***McKeeman*** and

Robinson fail to disclose all of the features of claim 24 (for at least the reasons discussed earlier), these references likewise fail to teach the features of dependent claim 30. For at least this reason, claim 30 is allowable.

Claim 33

Claim 33 is rejected under 35 U.S.C. 103(a) as being unpatentable over ***McKeeman*** and ***Robinson*** and further in view of ***Callahan, II.*** Applicants respectfully traverse this rejection.

Claim 33 depends indirectly from independent claim 24. Because ***McKeeman*** and ***Robinson*** fail to disclose all of the features of claim 24 (for at least the reasons discussed earlier), these references likewise fail to teach the features of dependent claim 33. For at least this reason, claim 33 is allowable.

Arguments with respect to other dependent claims have been noted. However, in view of the aforementioned arguments, these arguments are moot and, therefore, not specifically addressed. To the extent that characterizations of the prior art references or Applicants' claimed subject matter are not specifically addressed, it is to be understood that Applicants do not acquiesce to such characterization.

In view of the foregoing, it is respectfully submitted that all pending claims are in condition for immediate allowance. The Examiner is invited to contact the undersigned attorney at (713) 934-4069 with any questions, comments or suggestions relating to the referenced patent application.

Respectfully submitted,

WILLIAMS, MORGAN & AMERSON, P.C.
CUSTOMER NO. 62293

Date: November 17, 2010

By: /Jaison C. John/
Jaison C. John, Reg. No. 50,737
10333 Richmond, Suite 1100
Houston, Texas 77042
(713) 934-4069
(713) 934-7011 (facsimile)
ATTORNEY FOR APPLICANT(S)